Language Interoperability with Babel

Scott Kohn
with
Tammy Dahlgren, Tom Epperly, and Gary Kumfert

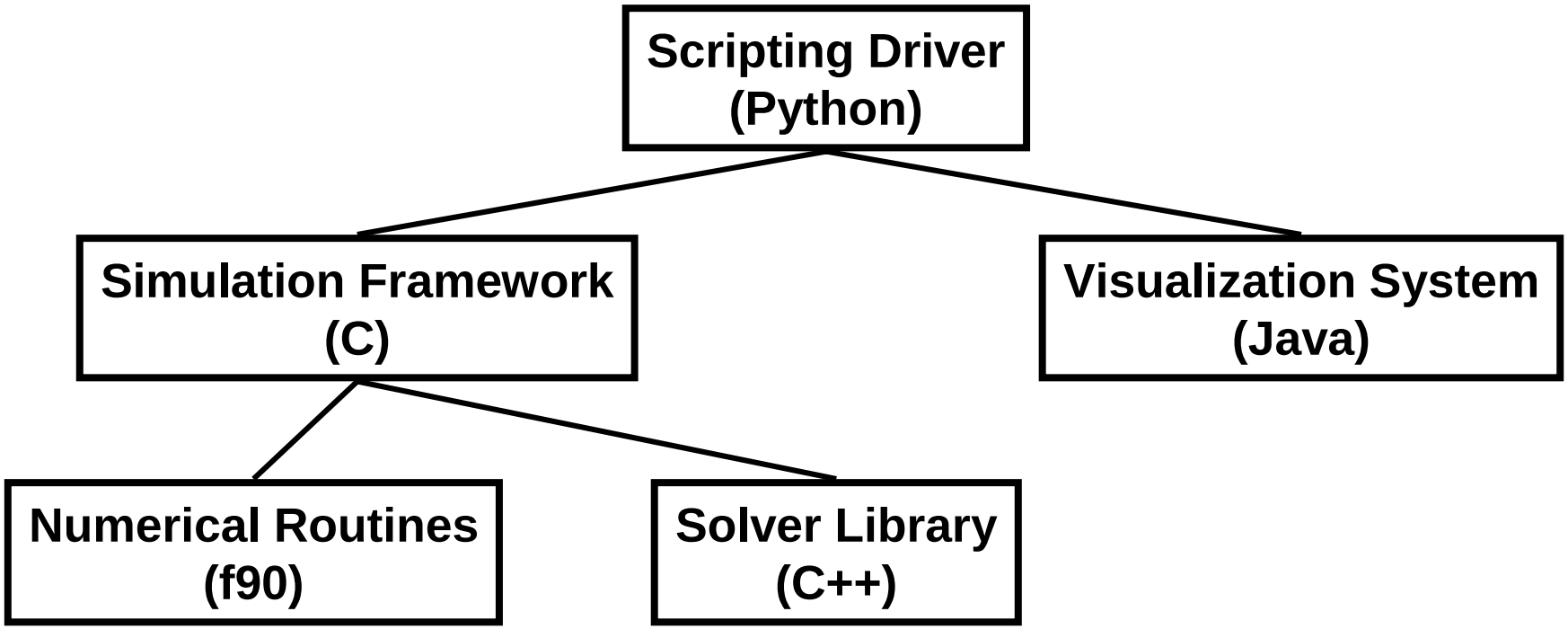
Center for Applied Scientific Computing
Lawrence Livermore National Laboratory

October 10, 2001

Overview

- Why you should care about language interoperability
- What Babel is and how it works
- “Hello World” tutorial
- Future work and contact information

DOE computational scientists use many different programming languages



ACTS libraries use Fortran, C, and C++

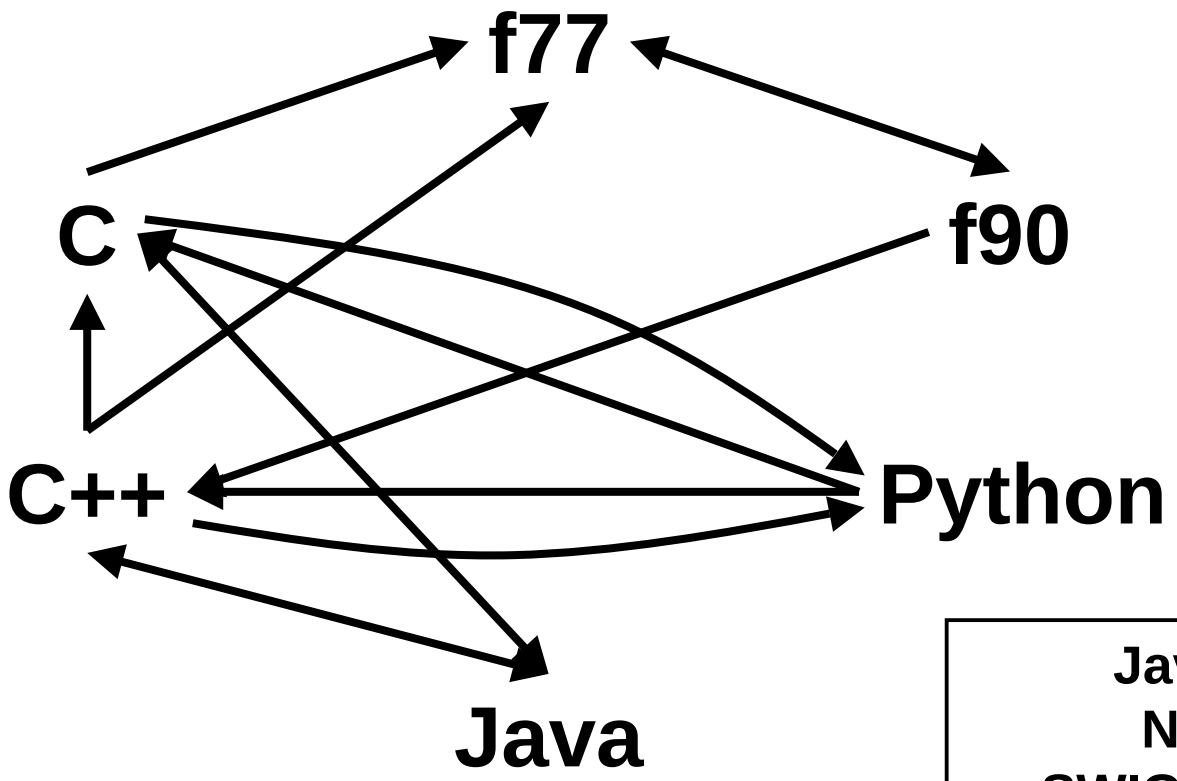
Why you should care about language interoperability

As a library developer, you have customers who are developing software in different languages (and you don't want to develop the language support by hand!)

As an application developer, you need to combine scientific libraries written in different languages

As a researcher, you would like to prototype a new preconditioner in a "high-level" language like Python or MATLAB but still call existing scientific solver libraries

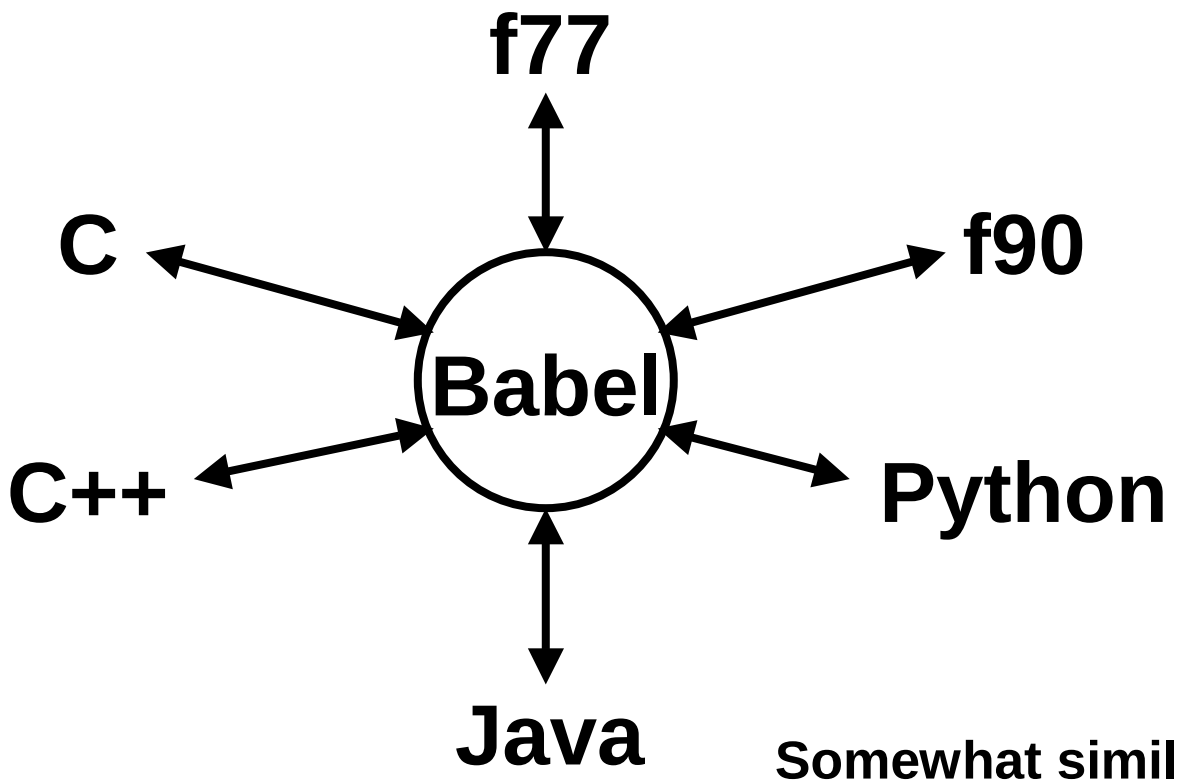
Existing language interoperability approaches are “point-to-point” solutions



Arrows indicate direction of supported function calls

Java JNI
Native
SWIG/SILOON
Platform Dependent
Python Library

Babel provides a unified approach in which all languages are considered peers



Somewhat similar to the CORBA approach in the business domain

There are many tradeoffs when choosing a language interoperability approach

- **Babel may not be the best solution for your problem**
- **SILOON or SWIG are outstanding tools**
  - generate glue code to wrap existing C or C++ libraries
  - mostly automatic; some minor annotations may be necessary
  - however, languages are not peers (e.g., C++ cannot call Python)
- **Babel supports a "peer" model but requires more effort**
  - library developers must write a separate interface definition file
  - library developers manually merge library with Babel glue code
  - Babel limits some language constructs (e.g., C++ templates)

Overview

- Why you should care about language interoperability
- ➨ What Babel is and how it works
- “Hello World” tutorial
- Future work and contact information

Babel design goals

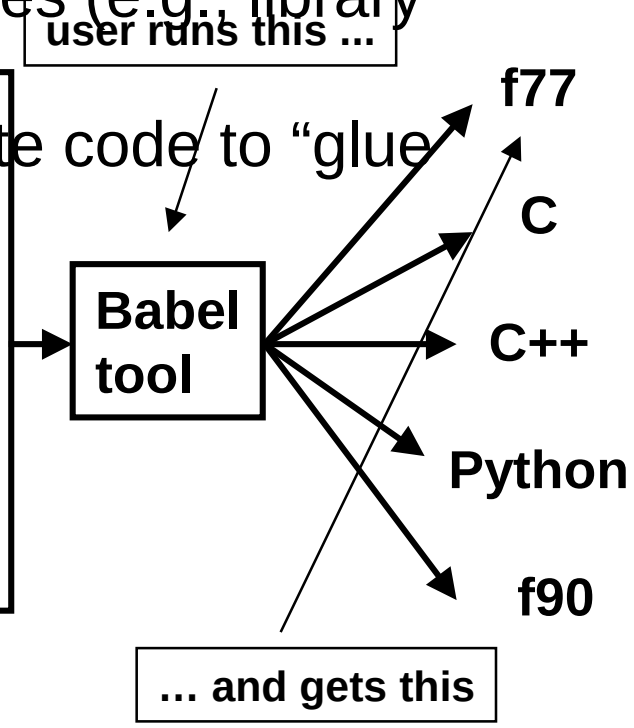
- **Mix Fortran, C, C++, Python, and Java as *peers***
- **Initial focus on same address-space performance**
- **Provide features unique to scientific libraries**
  - complex numbers and dense multi-dimensional arrays
  - support parallel data redistribution for distributed objects
- **Provide common useful run-time capabilities**
  - reference counting and dynamic casting
  - multi-language exceptions
  - remote procedure calls for distributed software

Babel uses a SIDL interface description of a library to generate glue code

- **SIDL is a "scientific" interface definition language**
  - we modified industry IDL technology for the scientific domain
  - SIDL describes calling interfaces (e.g., library specification)
  - our tools automatically generate code to "glue" languages

```
package Hypre {
   interface Vector {
      void axpy(in Vector x, in double a);
      double dot(in Vector x);
      …
   };
   interface Matrix {
      …
   };
};
```

library writer develops this



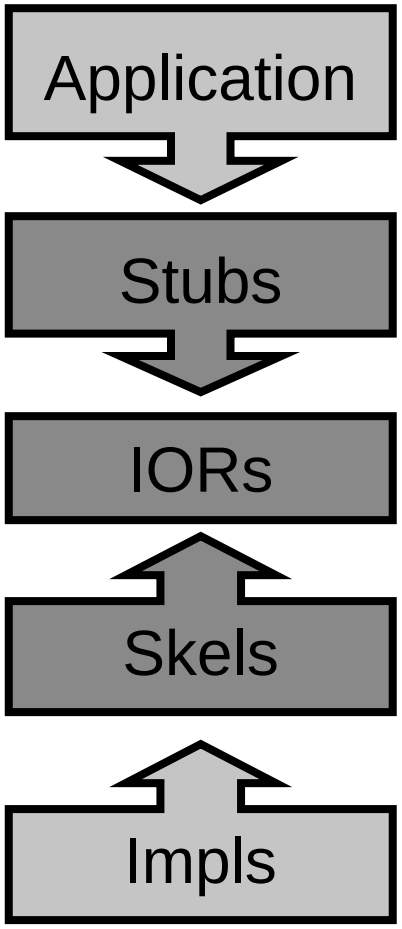
Short SIDL example from a collaboration with the *hypre* linear solvers team

```
version Hypre 1.0;

package Hypre {
   interface Vector {
      int copy( in Vector x );
      int dot( in Vector x, out double d );
      int axpy( in double a, in Vector x );
   }

   interface Solver extends LinearOperator {
      int apply( in Vector b, out Vector x );
      int getResidual( out Vector resid );
   }

   class StructVector implements-all Vector {
      void init( in StructGrid grid );
      int getNumGhost( out array<int> values );
   }

   class StructJacobi implements-all Solver {
      int setParameterDouble( in string name, in double value );
      int setup( in LinearOperator A, in Vector b, in Vector x );
   }
}
```

Babel's SIDL combines ideas from CORBA's IDL and Java

- **Java-like inheritance with classes and interfaces**
- **Methods may be static, final, or abstract**
- **Methods support these return and argument types:**
  - bool, char, int, long, opaque, string
  - dcomplex, double, fcomplex, float
  - any user-defined interface, class, or enumerated type
  - dense multi-dimensional arrays of the above types
- **Method arguments must be labeled `in`, `inout`, or `out`**
- **Optional `throws` clause for method exceptions**

Babel generates stubs, skeletons, and an intermediate object representation (IOR)

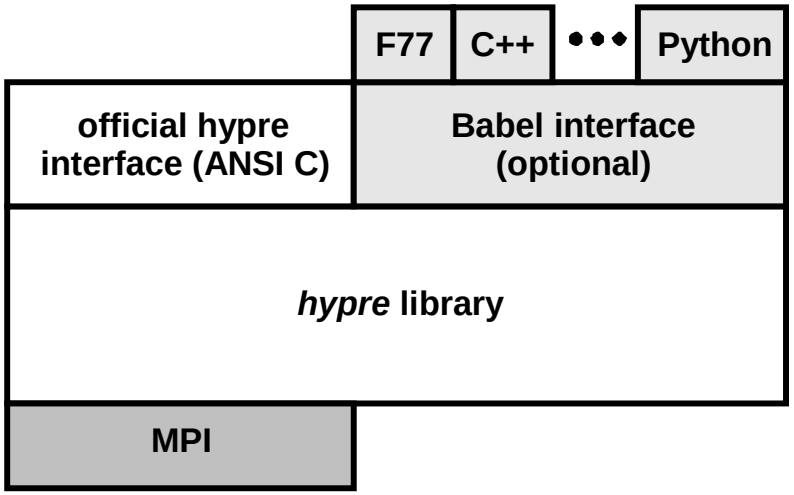


- **Application:** uses software in user's language of choice
- **Client Side Stubs:** translate from application language to IOR
- **Internal Object Representation**
- **Server Side Skeletons:** translate from IOR to implementation language
- **Implementation:** library developers' language of choice

We used SIDL technology to explore design issues for the *hypre* library

- ***hypre* supplies solver technology to DOE applications**
- **We generated a separate Babel interface for *hypre***
  - *hypre* designers explored new design approaches using our tools
  - automatic bindings for Fortran and other languages
  - parallel performance overheads too small to measure
  - improved SIDL and **Babel** based on feedback from *hypre*

S. Kohn, G. Kumfert, J. Painter, and C. Ribbens. "Divorcing Language Dependencies from a Scientific Software Library," *Proceedings of the SIAM Conference on Parallel Processing for Scientific Computing*, 2001



Current status of Babel (v0.6)

- **Language support**
  - — f77, C, C++, and Python “finished”
  - — Java client “finished” (except for arrays of user-defined objects)
  - — future language support: f90 and MATLAB
- **Regression suite checks over 3000 different cases**
- **Run-time support**
  - — safe dynamic casts and cross-language reference counting
  - — multi-language exceptions (e.g., throw in C++, catch in Java)
  - — dynamic loading similar to Java’s `Class.forName()`

http://www.llnl.gov/CASC/components

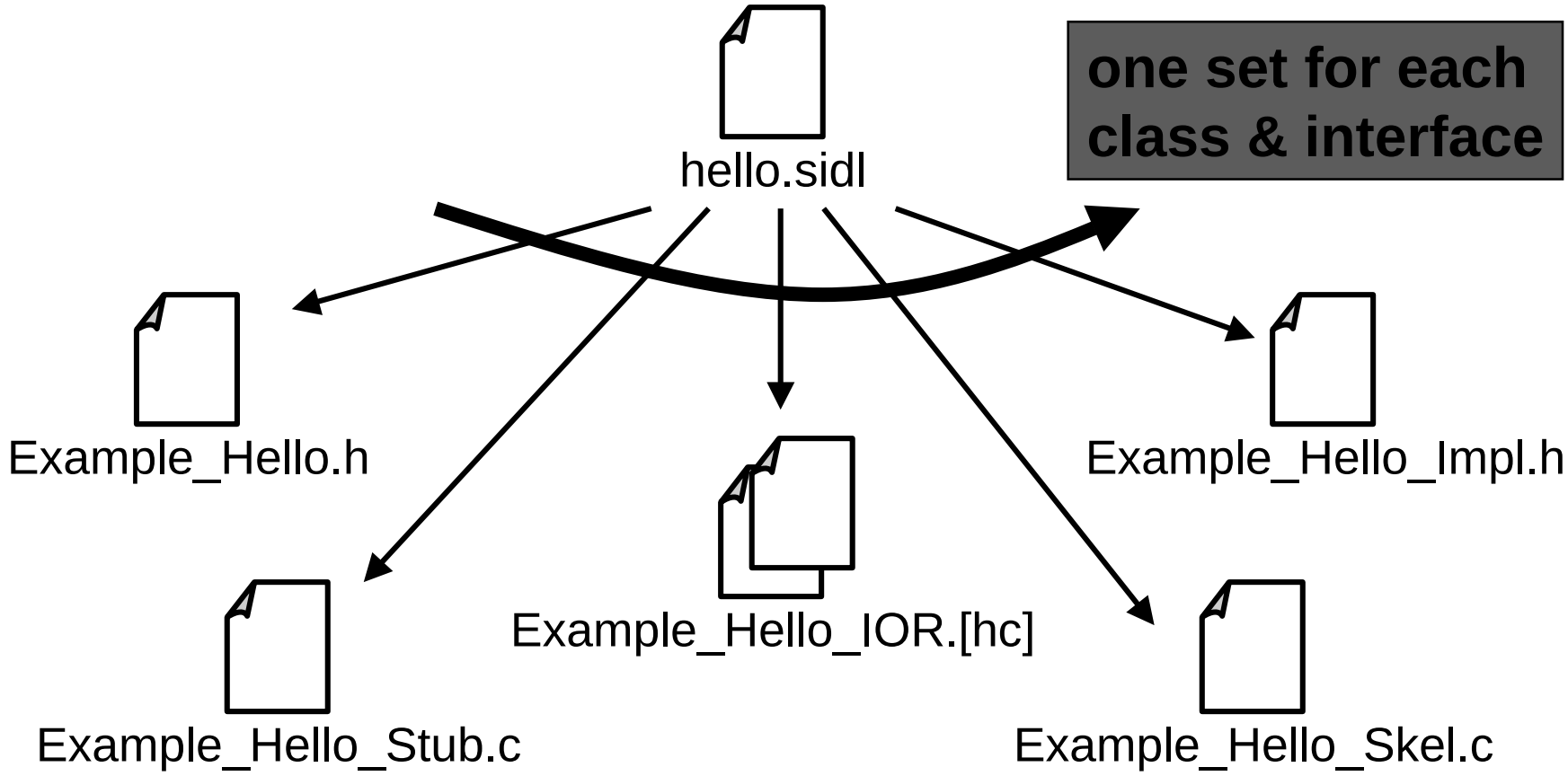
Overview

- Why you should care about language interoperability
- What Babel is and how it works
- ➨ “Hello World” tutorial
- Future work and contact information

Hello World:
SIDL input file

```
// hello.sidl
version Example 1.0;

package Example {
  class Hello {
    string getMsg();
  }
}
```

Hello World:
Babel generates a number of source files



Hello World: C implementation

Generated by Babel

```
/* Example_Hello_Impl.c */

/* Omit constructor and destructor function bodies */

char*
impl_Example_Hello_getMsg(
  Example_Hello self)
{
  /* DO-NOT-DELETE splicer.begin(Example.Hello.getMsg) */
  return strdup("Hello, world!");
  /* DO-NOT-DELETE splicer.end(Example.Hello.getMsg) */
}
```

Method implementation

Code between "splicer" comments saved during code re-generation

Hello World:
Fortran 77 implementation

```
C     Example_Hello_Impl.f

C     Omit constructor and destructor function bodies

      subroutine Example_Hello_getMsg_impl(self, retval)
      integer *8 self
      character *(*) retval

C     DO-NOT-DELETE splicer.begin(Example.Hello.getMsg)
      retval = 'Hello, world!'
C     DO-NOT-DELETE splicer.end(Example.Hello.getMsg)
      end
```

Note: return value currently at end of argument list to simplify Fortran portability

Hello World: C++ implementation

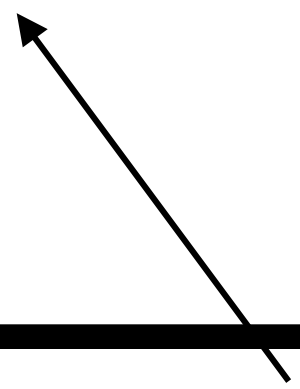
```
// Example_Hello_Impl.cc

// Omit constructor and destructor function bodies

string
Example::Hello_impl::getMsg() throw()
{
  // DO-NOT-DELETE splicer.begin(Example.Hello.getMsg)
  return string("Hello, world!");
  // DO-NOT-DELETE splicer.end(Example.Hello.getMsg)
}
```

Hello World: Python implementation

```
# Example/Hello_Impl.py

# Omit constructor and destructor function bodies

class Hello:

  def getMsg(self):
    # DO-NOT-DELETE splicer.begin(Example.Hello.getMsg)
    return “Hello, World!”
    # DO-NOT-DELETE splicer.end(Example.Hello.getMsg)
```

Hello World:
C client

```
/* main.c */
#include <stdio.h>
#include “Example_Hello.h”

int main(int argc, char** argv)
{
  Example_Hello h = Example_Hello__create( );
  char* msg = Example_Hello_getMsg(h);
  Example_Hello_deleteReference(h);

  printf(“%s\n”, msg );
  free(msg);

  return 0;
}
```



Create and destroy “hello” object

Hello World: Fortran 77 client

```
C     main.f

      program main
      integer *8 h
      character *32 msg

      call Example_Hello__create_f(h)
      call Example_Hello_getMsg_f(h, msg)
      call Example_Hello_deleteReference_f(h)

      print *, msg

      return
      end
```

Hello World: C++ client

```
// main.cc
#include <iostream>
#include “Example_Hello.hh”

int main(int argc, char** argv)
{
  Example::Hello h = Example::Hello::_create( );
  string msg = h.getMsg( );
  std::cout << msg << endl;
}
```

No explicit deallocation in C++ since reference counting and type conversion are managed by smart pointers

Hello World: Python client

```
# main.py
import Example.Hello

if __name__ == '__main__':
  h = Example.Hello.Hello( )
  print h.getMsg( )
```

Hello World: Java client

```
// HelloClient.java

public class HelloClient {
  public static void main(String args[]) {

    try {
      Example.Hello h = new Example.Hello( );
      String msg = h.getMsg( );
      System.out.println(msg);

    } catch (Throwable ex) {
      System.err.println(ex.toString());
    }
  }
}
```

Overview

- Why you should care about language interoperability
- What Babel is and how it works
- “Hello World” tutorial
- ➨ Future work and contact information

Future research directions

- **Add language support for Fortran 90 and MATLAB (?)**
- **Investigate component semantic descriptions in SIDL**
  - add constructs that describe “behavior” of classes
  - automatically generate simple run-time assertions
- **Support distributed communication**
  - remote procedure calls (like CORBA or Java RMI)
  - parallel data redistribution (M processors to N processors)
- **Possibly integrate Babel and SILOON technology**
- **Work with other library and application groups**

Contact information

- **Project web site**

  http://www.llnl.gov/CASC/components

- **Bug database web site**

  http://www-casc.llnl.gov/bugs

- **Project mail alias**

  components@llnl.gov or skohn@llnl.gov

- **Mailing lists**

  babel-announce@llnl.gov and babel-users@llnl.gov

Work performed under the auspices of the U. S. Department of Energy by the University of California, Lawrence Livermore National Laboratory under Contract W-7405-Eng-48.